Ground rules for the lectures

1. Engage…participate…ask…answer
2. Use your laptop / tablet / mobile (silent!!) to participate
3. If something's wrong, say so
   (e.g. you can't hear, others are distracting you,…)
4. Be aware that I have some hearing loss

CSN08114/CSN08414

Scripting for Cybersecurity and Networks

Lecture 1: Introduction to Python; booleans, numbers and strings

If statements

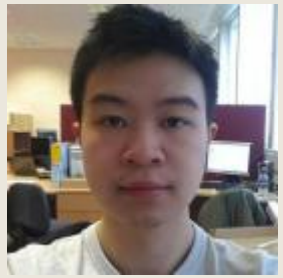
Module overview

Teaching team
Module organisation
Assessments

Teaching Team

Petra Leimich (Module leader)
p.leimich@napier.ac.uk
Room C.49

Sean McKeown
s.mckeown@napier.ac.uk



Owen Lo
o.lo@napier.ac.uk

And a big THANK YOU **to Rich Macfarlane** whose material this module is heavily based on!

Module outline

- 12 Taught Lectures + 12 Practical Labs

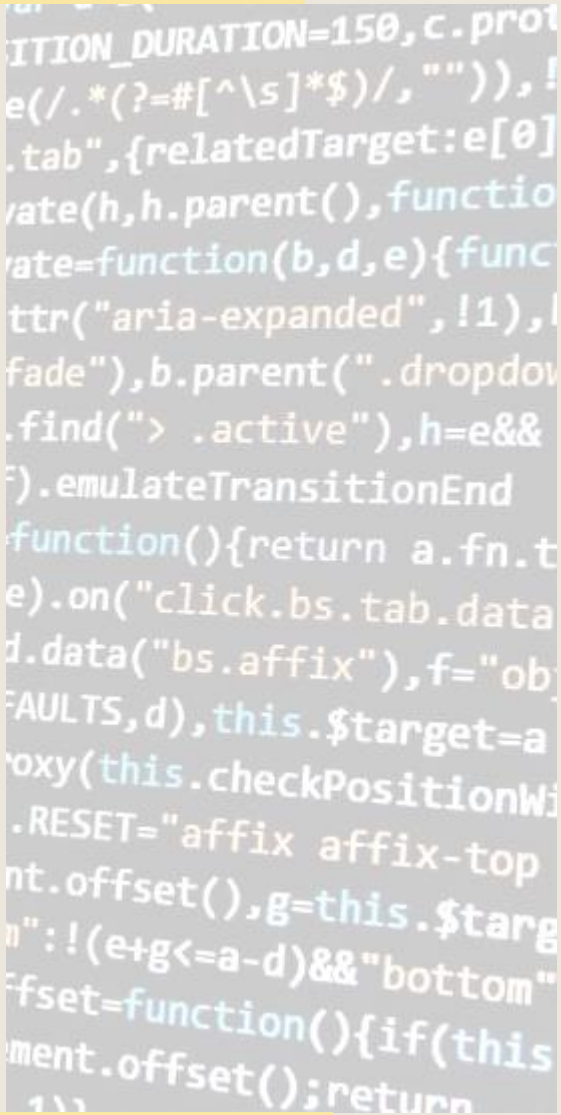
- **Lectures** - mix of talking about the language/theory + activities/demo of syntax/ Python examples + interactive quizzes
- **Labs** – examples of the theory from lectures + build approx. 2 Python scripts per lab
  - Scripts used to build coursework
  - Must complete labs in your own time! – 4+ hours/wk
- Labs pretty detailed, but do make notes of the answers and how you got them!
- Use Lecture notes + Moodle resources + online resources as reference in labs
- Can learn a lot of Python in one module!

Prerequisites

- Assumes that you have done some programming before
- But probably in another language

You should be familiar with, for example

- for loops
- if statements
- functions
- The idea of debugging



Assessment and feedback

Lab exercises

- Formative assessment and feedback
- Discuss your answers with us during the lab
- Selected exercises create scripts that will be part of the coursework

Class test

- 50%
- about half-way (week 8)
- Open book; during lab; 1.5 hours

Final coursework

- 50%
- A larger Case Study
- Submit and demo at the end of the module
- Heavily based on selected lab exercises

Module content

- Each topic starts with a lecture and continues with the lab the same week

week	Draft content
2	Scripting Languages/Python Overview, Environment - IDLE, Python syntax rules, Flow Control: If statements, For loops, Functions, Numbers, Booleans, arithmetic. Lab: Python Cryptanalysis - Keyspace calulator
3	Data Types: Strings & string formatting, Lists. Variables & Objects. Lab: encryption/decryption (Caesar Cipher)
4	Data Types: Tuples, Dictionaries; Modules/Scripts, Code Development: Exception Handling. Lab: Python Hashing/Password Cracking
5	Algorithms & complexity; External Data - reading and writing Files. Lab: Python Forensics Intro
6	Complexity & tuning
7	Statistics

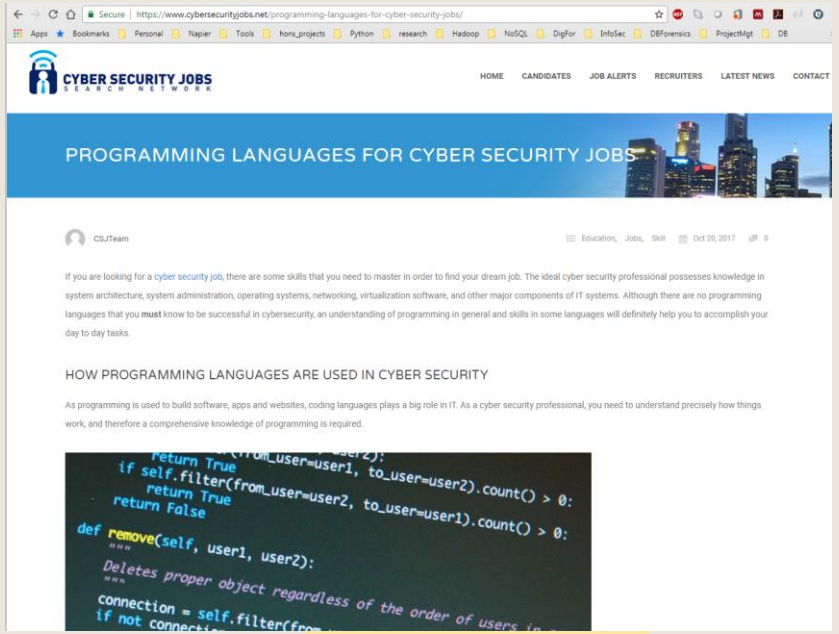
Module content

- Each topic starts with a lecture and continues with the lab the following week

week	Draft content
8	Text manipulation, System Programming, More External Data – Files/Web, Lab: Python Encryption
9	Regular Expressions
10	Computer networking with Python
11	Geolocation. Case study: mapping network traffic from IP addresses
12	Graph & Network Theory. Mapping code dependencies in Python scripts
13	CW Q&A

Why are we learning Python?

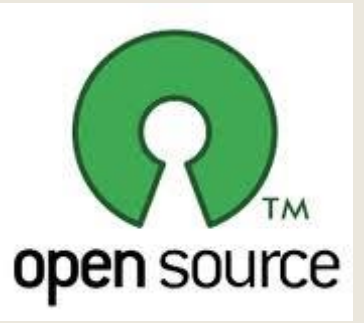
- Cybersecurity pros need programming skills
  - *Automate repetitive tasks*
  - *Analyse software for vulnerabilities*
  - *Identify malicious software*
  - *Script your own tools – e.g. plug ins for forensic software*
- Python is possibly the most popular language for cybersecurity
  - *Can be used for wide range of applications*
  - *Open source*
  - *Easy to learn*
  - *Rich set of libraries/tools available*
  - *Platform independent*



https://www.cybersecurityjobs.net/programming-languages-for-cyber-security-jobs/

Why learn Python?

- Designed to help produce Good Quality Code
  - *Readable/consistent/minimal code compared to other scripting languages – more readable than Perl*
  - *Easy to learn and use – easier than C, C++ ,C#, Java*
- Productivity
  - *Rapid Prototyping/Proof of concept - hacking*
  - *Less code to type/no compile stage*
- Portability
  - *Multi platform without any changes*
  - *Linux/Mac/Windows/Mobile*



Python Features

- Powerful but Light High Level Language - scripting and general purpose
- Rich set of libraries/tools
- Interpreted Language
  - *No compile phase/quick turnaround & prototyping*
- Dynamically Typed
  - *Typing at runtime - no need to pre-declare variables*
- Strongly Typed
  - *Dynamic type checking – reports misuse*
- Automated Memory Management
  - *Garbage collection*
- Simplicity of Scripting – procedural
- Object Oriented - optional

Version 3.x (currently 3.7)

- Most libraries now support 3.x
- Most textbooks and tutorial sites are for 3.x
- Eliminates old quirks that confuse learners
- Full Unicode support
  - *Clear distinction between strings and binary*
- New for 3.6:
  - *f-string formatting*
  - *secrets module*
  - *Underscores allowed in variable names*
  - *Type hints for named variables*

Version 2.7 - legacy

- Support will be discontinued 1/1/2020
- Still used by some existing applications
- Still default in some OS

We will use **Python 3.7** in this module

https://wiki.python.org/moin/Python2orPython3

http://python-notes.curiousefficiency.org/en/latest/python3/questions_and_answers.html#why-is-python-3-considered-a-better-language-to-teach-beginning-programmers

Running Python Code

IDLE and Command line
How do we create Python code

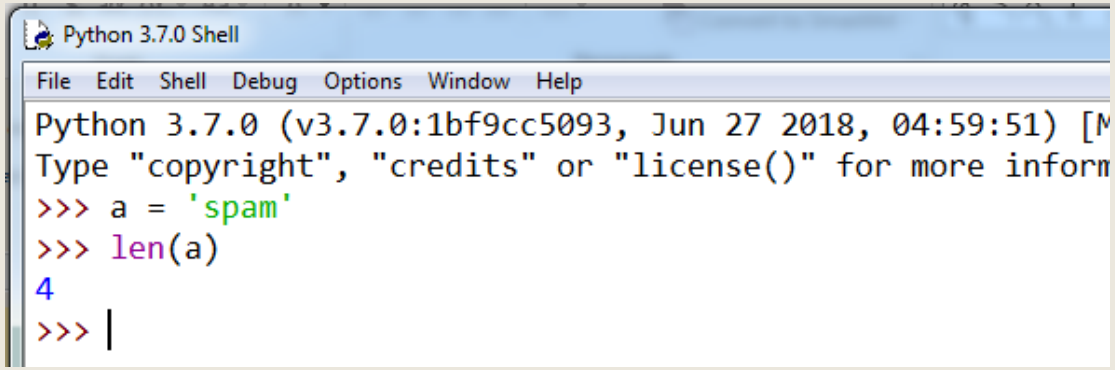
Running Python Code (demo in lab!)

IDLE - interactively

IDLE - run saved scripts

Command line - interactively

Command line - saved scripts




```
F:\Dropbox\CSN08114 Python>python hello.py
hello world
hello Petra
```

You should watch the recordings of
Lecture 00a Python Background and
Lecture 00b Running Python in Windows
during independent study.

Go to www.menti.com

code **93 01 25**

Variables & Objects

Python Objects
Python Built-in Object Types
Dynamic Typing

Python Variables (objects)

- Variables allow us to store data
- No need to declare variable or its data type
- Assign a value with = operator
  - *Automatically created/typed, memory assigned*

```
>>> a = 6                  # create integer object
>>> b = "hello"            # create string object
>>> f = open("x.txt")      # create file object
```

- Variables are Objects. The assignment defines the type of Object created (**dynamically typed**)
- Variables are case sensitive (like everything in Python)

```
>>> A

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    A
NameError: name 'A' is not defined
```

Python Variables

```
>>> int1 = 5
>>> flt = 0.945
>>> txt = 'Hello!'
>>> int1/flt
5.291005291005291
>>> txt*2
'Hello!Hello!'
```

Variables are assigned values without pre-defining a type

```
>>> m,n=5,8
>>> m*n
40
>>> x = y = z = "string value"
>>> x
'string value'
>>> z
'string value'
```

Several variables can be assigned values in one statement

Python stops with useful error (some languages do not)

```
>>> print(int1)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(int1)
NameError: name 'int1' is not defined
```

Variables must be assigned a value before use

Object Data Types we will learn about:

- Numbers: Integer, Float
- Boolean
- String
- Collections: List, Dictionary, Tuple
- File
- Module, Function - program units are objects too

- Can create very advanced applications with only the built-in Object data types

Python Variables

- To create **Python Programs** we store code in **Modules/Functions**
- Functions contain **Statements (& Expressions)**
- Statements (& Expressions) can create/manipulate **Python Variables**

- Python Variables are all **Objects**

- We will use **Python Built-in Objects**
- OOP – create your own Python Objects

Syntax / Flow Control

And Booleans

Python statements
Differences to other languages - What Python has removed!
Syntax for blocks of code
Booleans, comparison operators
If (conditional) statements
Comments

Python Code Syntax

Python Statement

Line of Python code to do something – logic of program

*Made up of Python **Expressions***

Expressions create and manipulate Python Objects

```
>>> if x == 5: print ('Python is great')
```

No Statement termination character

Python has no statement termination character (often a semi-colon)

Uses end of line instead – less typing!

\ backslash can be used to continue lines:

```
>>> z = x \
... + y
```

```
>>> y = 2
>>> x = 5
>>> x*y
10
>>> _+9
19
```

Assignment statements create variables and associated objects

_ (underscore) is last printed value

Scripting for Cybersec & Networks

Blocks and indentation

Indentation is used to group together a 'block' of code

```
for msg in ('all','you','good morning'):
    print ("hello " + msg)
    print ("hello again " + msg)
```

indentation

Compound Statement
- Embedded statements

Less cluttered than other languages *(which often use brackets)*

Colon ':' *added to show that a code block/indentation is expected*

You will forget this many times!!

4 spaces commonly used for indentation *(2 – Google)*

DO NOT mix tabs and spaces!

Spot the errors

```
if True:
    print ("true")
    print ("do")
else:
    print ("false")
    print ("do not")
print ("more code")
```

Don't have to type **brackets** round condition, **{}** around code blocks, or **;** at end of line

```
if True
    print ("true")
    print "do"
else:
    print ("false")
     print ("do not")
print ("more code")
```

Missing colon :

Missing brackets

- Inconsistent indentation produces Runtime Error
- Indentation must be consistent
- Unlike C style languages!

Scripting for Cybersec & Networks

Another IF statement example

```
if speed >= 70:
    print ("Driving License please")
    if mood == 'terrible' or speed >= 100:
        print ('You are in big trouble!')
    elif mood == 'bad' or speed >= 90:
        print ("6 points for you")

    elif mood == 'ok' or speed >= 80:
        print ("You are getting a ticket")

    else:
        print ("Let's try to keep the speed down from now on")
```

Don't forget the **:**

Multiple conditions
and/or/not spelt out

Multiple conditions
Using elif (no switch/case)

Use indents to group blocks

Use single quotes inside double or vice versa

IF statement syntax

- **Why do we need if statements?**
  - *Decides which code to run*
  - *Based on Boolean test being True or False*

- **syntax:**

```
if <boolean test>:
      <indented code block>
elif <boolean test>:
     <indented code block>
else:
     <indented code block>
```

Boolean Expression
Expression which equates to True or False

Booleans

Boolean Data Type

Values: `True, False`

```
b = True
b
True
b == False
False
```

So what's the difference between = and == ?

Boolean Operators: `and, or, not`

```
b1 = True
b2 = False
b1 and b2
False
```

`True and True`	`True`
`True and False`	`False`
`False and False`	`False`

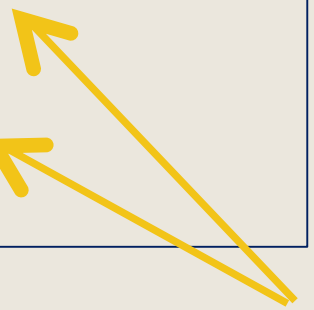
Comparison operators

Compare two values and evaluate to **True** or **False**

`==`	*Equal to*
`!=`	*Not Equal*
`<>`	*Not Equal*
`<`	*Less than*
`>`	*Greater than*
`<=`	*Less than or equal to*
`>=`	*Greater than or equal to*

```
l = len('hack')

l == 4
True
l >= 8
False
```



Boolean Expression
equates to True or False

Comments

Hash # used for single line comments

Everything to end of line ignored

End of line is end of comment

```
#
# Script:  Hello World
# Created: Sept 2017 by Petra
#
print ("Hello World")    # Print the hello msg

""" triple quotes can be used for comments that span
    more than one line """
```

Scripts, functions, and introducing modules

Python Functions
Python BIFs (built in functions)
Python Modules part 1
Scripts

Python Code Syntax

Indentation is used to group together a 'block' of code

```
def print_msg(msg):
    print ("hello " + msg)
    print ("hello again " + msg)
```

indentation

Compound Statement
- Embedded statements

Less cluttered than other languages *(which often use brackets)*

Colon ':' *added to show that a code block/indentation is expected*

You will forget this many times!!

4 spaces commonly used for indentation *(2 – Google)*

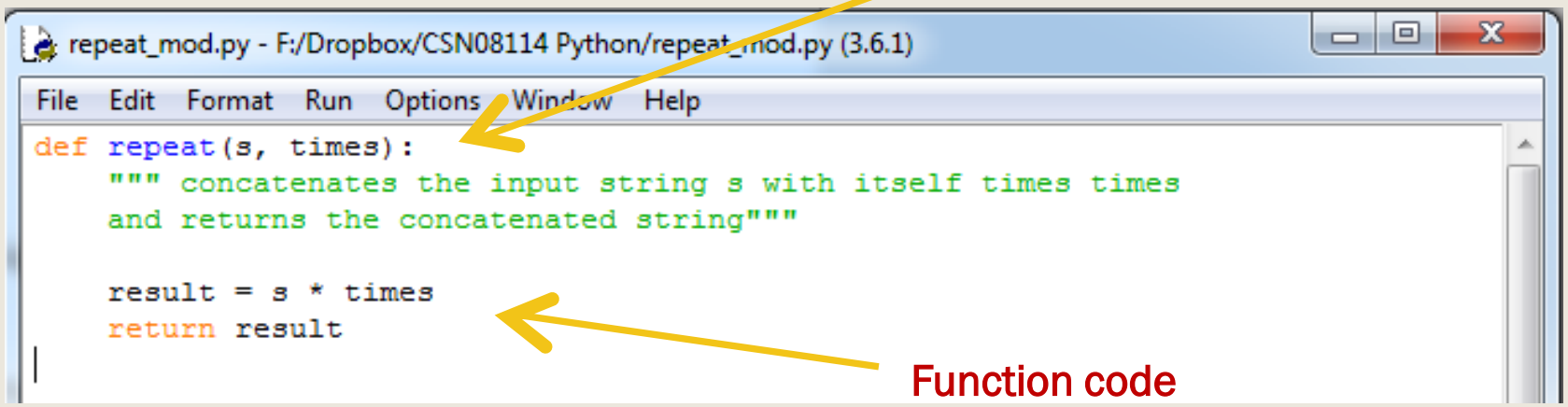
DO NOT mix tabs and spaces!

Python Functions

- Reuse code using Python **Functions**

Function Syntax:

```
def <function_name>:
        <indented code block>
```

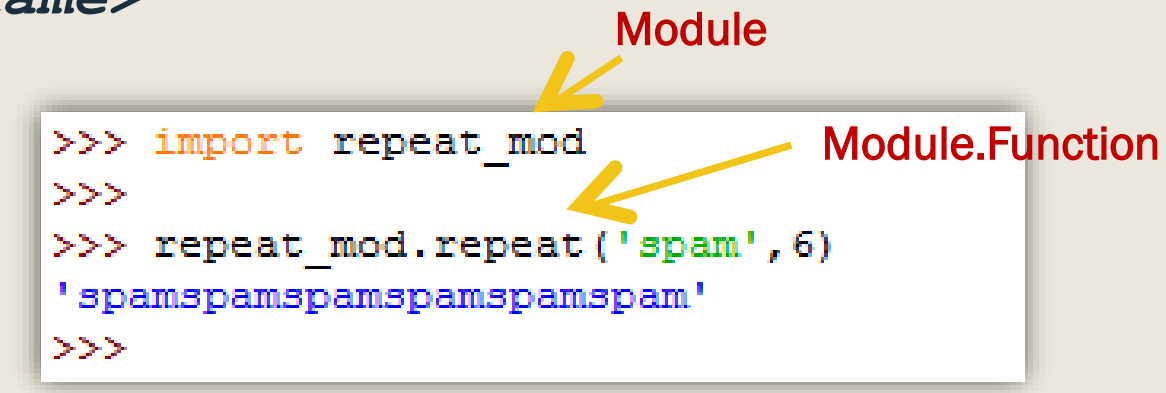


- a **Module** is a file that contains functions

Using Functions

To use a function outside of the module where it is defined, you need to import the module.

`import <module_name>`



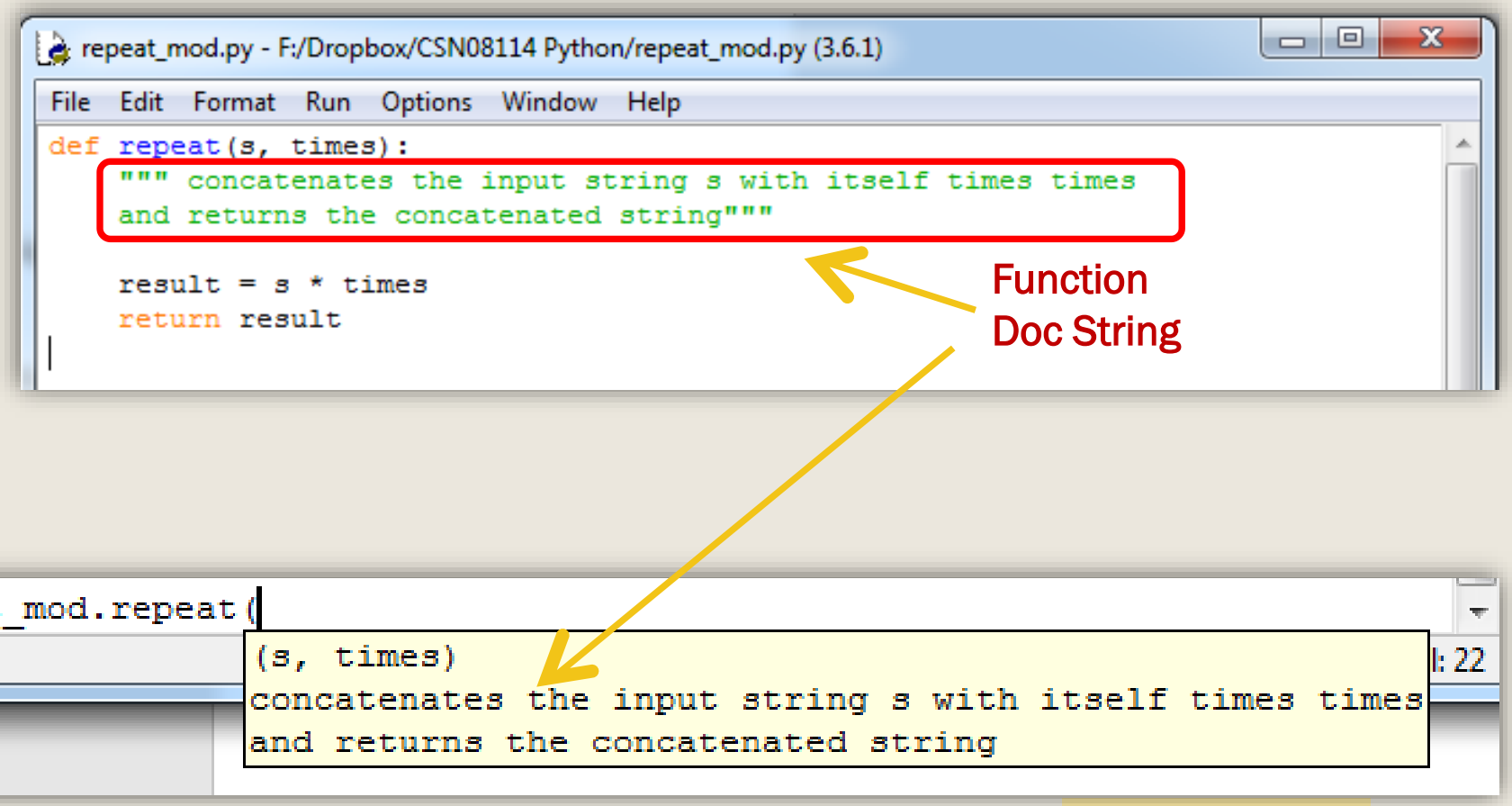
If you need only one function from a big module, you can import only the function:

`from <module_name> import <function_name>`

- On import, all code in the .py file is run – function definitions are created.
- Functions can be run with **`module.function()`** syntax
- **`reload()`** can be used if changes are made to the file

The function doc string

- The function doc string is a comment right after the function definition
- It works like a help string
- Pops up when you use the function



The Python Standard Library

Contains

- Built-in **functions** (BIFs) (http://docs.python.org/library/functions.html)
- Built-in **modules** (https://docs.python.org/3/py-modindex.html)
- And many more (constants, types, exceptions, …)

- These are always available when the Python interpreter is run.

Python Built In Funtions (BIF)

Previous topic
1. Introduction

Next topic
3. Built-in Constants

This Page
Report a Bug
Show Source

2. Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Using built-in **functions** example: Checking type of object / variable

- Built-in functions can be used "right out of the box"
- E.g. You may want / need to check what type a variable is
- **Useful BIFs:**
  - *type(v)*
  - *isinstance(v, type)*

```
>>> v = 1.345
>>> x = "good afternoon"
>>> type(v)
<class 'float'>
>>> type(x)
<class 'str'>
```

```
>>> isinstance(v,int)
False
>>> isinstance(x,str)
True
```

Using standard library Modules

- To use functions from a standard library **Module**, you need to import it
- examples:

```
>>> import os
>>> os.getcwd()
'C:\\CSN08114 Python'

>>> from math import sqrt
>>> sqrt(100)
10
```

External (3rd party) Modules

- If the Standard Library does not contain suitable functions, there are many* Python modules/packages available in **PyPI – The Python Package Index**

  `https://pypi.org`

- Need to install these first before importing
- Use pip (in Windows) → more later

*13/07/2017: **112142** packages
*21/09/2017: **117516** packages
*10/09/2018: **151587** projects

is PyPI safe??

- No checks - anyone can upload
- Only developer can modify existing package
- pyto-squatting attack demonstrated in September 2017
- Github security alerts support Python projects since July 2018

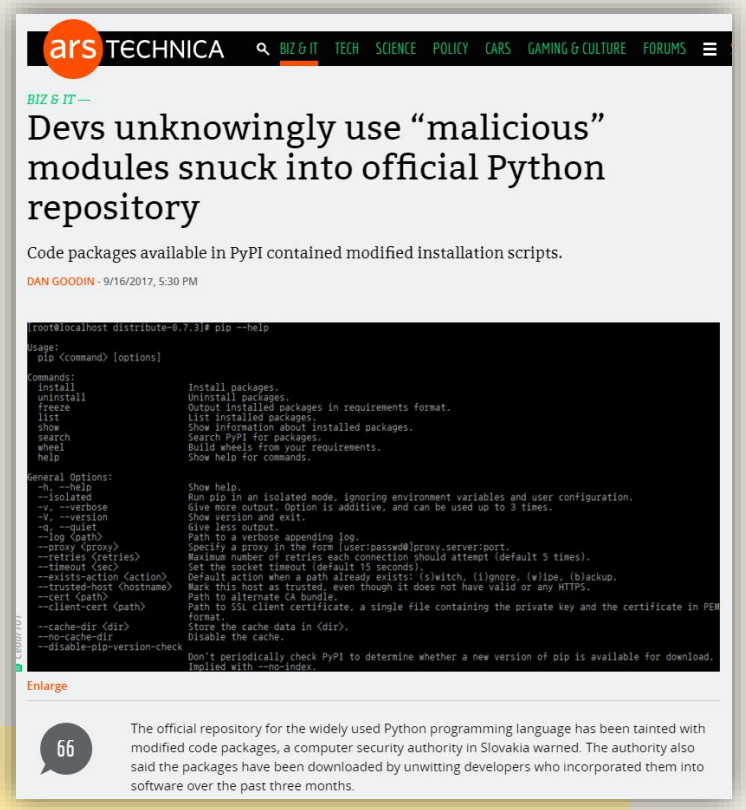
https://arstechnica.com/information-technology/2017/09/devs-unknowingly-use-malicious-modules-put-into-official-python-repository/

https://www.theregister.co.uk/2017/09/15/pretend_python_packages_prey_on_poor_typing/

https://www.pytosquatting.org/

https://www.bleepingcomputer.com/news/security/ten-malicious-libraries-found-on-pypi-python-package-index/

https://www.bleepingcomputer.com/news/security/github-security-alerts-now-support-python-projects/



Numbers

Python Number Objects
Number Object Types

Python Numbers

- 2 main numeric object types
  - **int** – Signed Integer
  - **float** - Floating Point Numbers
  - 0 and 1 are int but also work like Booleans (represent False and True respectively)

```
>>> var = 4.5
>>> type(var)
<class 'float'>
>>> var = 2
>>> type(var)
<class 'int'>
>>> var = 11111111111111111111111111111111111111
>>> type(var)
<class 'int'>
```

- Immutable
  - *Can't change*
  - *Changing value of a number actually produces a new number object*

Python Numbers

Creation Examples:

bool

```
bored = False
```

int

```
lecture_marks_out_of_ten_so_far = 9
longerint = -795875858954999
```

float

```
float1 = 3.1446546456
```

Useful BIFs to Check type of variable:

type()

isinstance()

these are useful with Python's dynamic typing

```
>>> b1=False
>>> b2=0
>>> b1==b2
True
>>> type(b1)
<class 'bool'>
>>> type(b2)
<class 'int'>
>>> isinstance(b2,float)
False
>>> var=3.1516
>>> type(var)
<class 'float'>
>>> isinstance(var,float)
True
>>> isinstance(var,int)
False
>>> isinstance(b2,bool)
False
```

Python Numeric Ops/Functions

- **Python Numeric Operators:**

  Addition, Subtraction: `+ -`

  Multiplication, Division, integer division: `* / //`

  Modulus: `%`

  x to the power of y: `x**y`

- **Numeric BIFs:**
  - *Conversion: int(), float(), str()*
- **Math** Module contains more advanced numeric functions
  - `math.sqrt(x)` *Square root of x*
  - `math.pow(x,y)` *x to the power y with scientific notation output*

```
>>> a = 10
>>> b = 3
>>> a/b
3.3333333333333335
>>> a//b
3
>>> a%b
1
>>> a*b
30
>>> a**b
1000
```

```
>>> import math
>>> math.pow(a,b)
1000.0
```

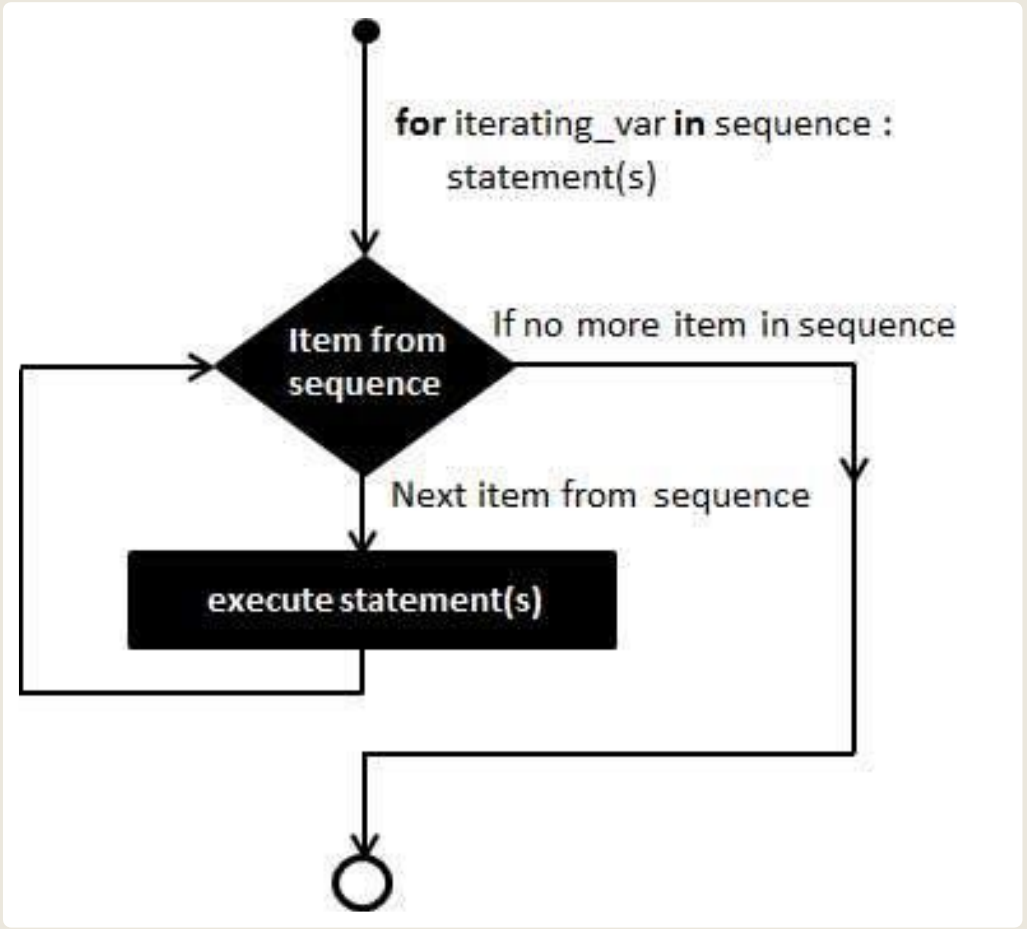
Flow control: FOR loops

Python FOR loops

- Execute a block a specified number of times

Basic syntax

```
for var in sequence:
    statement(s)
# move on when sequence finished
```



oop.jpg

FOR loop example

```
>>> sum=0
>>> for i in range(10):
        sum=sum+1
        print(f'iteration {i}: sum={sum:2d}')


iteration 0: sum= 1
iteration 1: sum= 2
iteration 2: sum= 3
iteration 3: sum= 4
iteration 4: sum= 5
iteration 5: sum= 6
iteration 6: sum= 7
iteration 7: sum= 8
iteration 8: sum= 9
iteration 9: sum=10
```

range() built in function generates sequences

Example of number formatting

Example of an f-string (next week)

Application: print keyspace for passwords length 1-10

```
>>> for passLen in range(1, 11):
        print (f'{passLen} {math.pow(95,passLen)}')
```

```
>>> import math
>>> for passLen in range(1, 11):
        print (f'{passLen} {math.pow(95,passLen)}')


1 95.0
2 9025.0
3 857375.0
4 81450625.0
5 7737809375.0
6 735091890625.0
7 69833729609375.0
8 6634204312890625.0
9 6.302494097246094e+17
10 5.987369392383789e+19
```

Python Loop – **break** statement

break

Jumps out of current loop, and onto next statement after loop:

```
>>> for passLen in range(1, 11):
        if math.pow(95,passLen) > 100000000:
            break
        print (f'{passLen} {math.pow(95,passLen)}')

1 95.0
2 9025.0
3 857375.0
4 81450625.0
>>>
```

Python Loop – **continue** statement

continue

Jumps out of current loop iteration,

and onto next iteration in the loop:

```
>>> for passLen in range(1, 11):
        print (f'iteration {passLen}:',end='')
        if math.pow(95,passLen) > 100000000:
            continue
        print (f'{math.pow(95,passLen)}')
```

```
iteration 1: 95.0
iteration 2: 9025.0
iteration 3: 857375.0
iteration 4: 81450625.0
iteration 5: iteration 6: iteration 7: iteration 8: iteration 9: iteration 10:
```

Scientific notation

A note on Numbers: Scientific Notation

SCIENTIFIC NOTATION

- used to handle very large or very small numbers more easily
- Precision is variable
- Split the actual digits from the power of 10 (the magnitude)
- Programming languages use E+x or E-x instead

Standard notation	Scientific notation	Python Scientific notation
12345	1.2345×10^{4}	1.2345e+4
937000099	9.37×10^{8}	9.37e+8
0.0000012345	1.2345×10^{-6}	1.2345e-6
0.009100099	9.1×10^{-3}	9.1e-3

- To convert easily, think how many places you need to move the decimal point!
- See https://www.mathsisfun.com/numbers/scientific-notation.html.

Scientific notation in Python

- e+x or e-x to handle very large or very small numbers.
- Used particularly with floats and with math module
- Precision is variable

```
>>> 123456789.0**3
1.8816763717891548e+24
>>> 123456789**3
188167637178915486089706 9
>>> import math
>>> math.pow(123456789,3)
1.8816763717891548e+24
```

e+24 means to **move the decimal point 24 digits to the right**

i.e. 188167637178915480000000 0
(note that we have lost precision)

Converting notation in Python

- floats do use scientific notation
- ints do not
- Can use float() and int() functions to "convert"

```
>>> import math

>>> v1= math.pow(123456789,3)
>>> v1
1.8816763717891548e+24
>>> type(v1)
<class 'float'>
>>> int(v1)
1881676371789154785165312
```

```
>>> v2= int(math.pow(123456789,3))
>>> v2
1881676371789154785165312
>>> type(v2)
<class 'int'>
>>> float(v2)
1.8816763717891548e+24

>>> v1==v2
True
```

Go to www.menti.com and use code **93 01 25**

Practical Lab

Code breaking / password recovery:
How long does it take to brute force a password?

Script1: Python Keyspace Calculator

- **Cryptanalysis – Code Breaking**

  How long does it take to Brute Force a Password??

- Create a script
  - *to calculate Password combinations (keyspace) for various length passwords*
  - *And from that the average time to brute force an ASCII password*

- **What would we need to know?**

Python Keyspace Calculator

- **Code Breaking - Brute Force**

- **Calculate Password Key Space - possible combinations**
  - ***Chars*** *= Number of characters in character set*
  - ***PassLen*** *= Number of characters in a password*

  - ***`Keyspace = Chars^PassLen`***

Python Keyspace Calculator

- ***Chars*** *= Number of characters in character set*
  - ASCII Characters?
    - ***95 Printable***

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
32	[space]	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	[backspace]

- ***PassLen*** *= password length (Number of characters in password)*
  - 1
  - 8
- ***`Keyspace = Chars**PassLen`***
  - **95**1 = 95**
  - **95**8 = 6,634,204,312,890,625**

Python Keyspace Calculator

- **Lab – run script**
  - *What do we need to add? Where?*

Add code to get_keyspace() to calculate the keyspace

But How?

keyspace_start.py - F:/Dropbox/CSN08114 Python/keyspace_start.py (3.6.1)

File Edit Format Run Options Window Help

```
# Script:  keyspace.py
# Desc:    Calculate keyspace/entropy of password
# Author:
# Created:

import sys, math

def get_keyspace(passwd):
        """prints the entropy value for an ascii password"""
        print ('[*] keyspace')
        char_set = 95
        keyspace = 0 # ... ADD YOUR CODE HERE ...
        print ('[*] Password:', passwd, '- Total of:', str(keyspace), 'key combinations')

# test case
passwd = 'test' # change the passwd variable to test
# call keyspace calc function
get_keyspace(passwd)
```

```
>>>
[*] keyspace
[*] Password: test - Total of: 0 key combinations
>>>
```

Python Keyspace Calculator

- **Calculate Average Attempts for a Key Space**
  - *How many attempts before password recovered?*
  - *Code breaking law of averages*
    – code has equal chance of being found anywhere in keyspace
    - Equal chance of being first or last combination in keyspace
    - On average found half way through key space
  - ***`AverageAttempts = Keyspace/2`***
    - **95/2 = 47.5 attempts on average to crack**
    - **6634204312890625 / 2 = 3317102156445312.5**

Python Keyspace Calculator

- **Calculate Average Time to crack Password**
  - *How much time before password recovered?*
  - *Depends on how many passwords system can try in a hour?*
    - Let's say **`PasswdsPerHour = 100,000,000`**

  - ***`AverageTimeToCrack = AvgAttempts/PasswdsPerHour`***
    - **47.5 / 100,000,000 = 0.000000475 hours**
      = 0.00171 secs *(*60 *60)*
    - **3.11 x10^85 / 100,000,000 = 66342043 hours**
      = 7573 years (/24 /365)

Python3 Documentation

Official Documentation:

http://www.python.org/doc/

Quick Reference:

http://docs.python.org/reference/

https://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/python32.pdf

Cheat Sheets:

https://perso.limsi.fr/pointal/_media/python:cours:mementopython3-english.pdf

http://sixthresearcher.com/wp-content/uploads/2016/12/Python3_reference_cheat_sheet.pdf

Tutorials:

https://www.sololearn.com/Course/Python (very interactive, mobile apps available, highly recommended)

https://www.tutorialspoint.com/python3/

http://www.python-course.eu/python3_course.php

https://dbader.org/blog/ (more tips and tricks)

If you find any good online resources please post them in the moodle discussion forum!